Lab 5 – Pointers and Arrays
The purpose of this lab is to practice using pointers to manipulate the data in arrays – in this case, in arrays of characters. We will be building functions that add entries to, print, and delete entries in an array of characters. In addition, we will begin separating our code into separate files and using make files to build our system.

The Part We Give You
In this lab, you are going to build two c files: runner.c that will contain a main function and charlist.c that will contain the functions that manipulate our arrays of characters. In order for runner.c to be able to use the functions in charlist.c, we need to specify their prototypes in an include file. So, start by creating a folder to hold the files for this lab and, in that folder, create charlist.h and make it look like this:

```
/***************************************************************************/
/* initialize the array pointed to by p so that it is ready to           */
/* be a character list                                                 */
/***************************************************************************/
void initialize_list(char *c);

/***************************************************************************/
/* print out the characters in the list that p points to                */
/***************************************************************************/
void print_list(char *c);

/***************************************************************************/
/* Add a character to the end of the list that p points to             */
/***************************************************************************/
void add_to_list(char *p, char x);
```

This specifies the functions that we will build in charlist.c. By including this file in our runner.c, we let the compiler know what the parameters and return types of the functions are so that it can build the appropriate code when our main function uses them.
In order to see how these functions are used, create a runner.c file and make it look like this:

```c
#include "charlist.h"
#include <stdio.h>

/***************************************************************/
/* A simple main function to play with the functionality in   */
/* charlist.c                                                */
/***************************************************************/
int main()
{
    char x[10];
    initialize_list(x);
    add_to_list(x, 'c');
    add_to_list(x, 'a');
    add_to_list(x, 't');
    printf("Expect to see cat: ");
    print_list(x);

    return 0;
}
```

You should be able to explain how this code

- creates an array of ten characters
- passes a pointer to that array into the initialize_list, add_to_list, and print_list functions, and
- includes the specifications of the functions in charlist.h.

While there are unit testing tools for c (similar to JUnit), their use is not well-spread. Instead, this main function encodes a test. It prints out what the result should look like and then prints the list. When you run it, you can compare the expected to the actual to see if the code is working properly.

The last piece of code we need are the functions in charlist.c:

```c
#include "charlist.h"
#include <stdio.h>

/***************************************************************/
/* initialize the array pointed to by p so that it is ready to   */
/* be a character list                                         */
/***************************************************************/
void initialize_list(char *p)
{
    *p = '\0';
}
```
/*****************************/
/* print out the characters in the list */
/*********************/
void print_list(char *p)
{
  char *temp = p;
  while (*temp != '\0')
  {
    printf("%c", *temp);
    temp++;
  }
  printf("\n");
}

/*********************/
/* Add a character to the end of the list */
/*********************/
void add_to_list(char *p, char x)
{
  char *temp = p;
  while (*temp != '\0')
  {
    temp++;
  }
  *temp = x;
  temp++;
  *temp = '\0';
}

There are a few things you should observer about this code:

- initialize_list just puts a special character (call NULL) as the first position in the array. We will use that character to detect the end of the data in the array since it isn’t always full.
- print_list is given a pointer to the beginning of the list. It uses another pointer (temp) to start at that position and walk through the list printing each character until it reaches the terminating NULL
- add_to_list is given a pointer to the beginning of the list and a character that is to be added to the list. It searches the array for the terminating NULL, puts the new character in that position and puts a new NULL after that position. Note that we are doing no error checking – we can add stuff way beyond the end of the array. This is very bad and could lead to corruption of memory which would crash our code. We must certainly learn to do better than this soon!

Now that we have all of the code, we need to build an executable from it. Since we have multiple files, hand running gcc could become complex, so we will use make instead. Create a file named makefile with the resulting executable file name as “charplay”. You can reuse makefile from Lab 3.

**The Part You Build (the easy case)**

Your task in this lab is to build a function that will delete one character from the array. Its declaration must look like this:
void delete_from_list(char *p);

Start by adding this declaration to charlist.h. In order to understand what this function must do, let’s add another “test” to our main function:

```c
    delete_from_list(&(x[2]));
    printf("Expect to see ca: ");
    print_list(x);
```

This code is using `delete_from_list` to remove the character in position 2 of the array (which is the ‘t’ in “cat”). Remember that the & operator gets the address of something (which is a pointer to that something). Since this test is removing the last character in the array, all you have to do to get it to pass is put the NULL terminator at the position pointed to by the parameter to the function. Don’t go on until the output is what the test expects.

### The Part You Build (for reals)
Removing something from the list is not so simple when we are removing something in the middle of the list. Since the print function stops at the first NULL terminator, we can’t just put a NULL in the middle of our data. Instead, we are going to have to move the data that is after what we are deleting one position to the left. For example, if our array looks like this:

```
A B C D \0 X
```

and we deleted the B, the array should end up looking like this:

```
A C D \0 \0 X
```

Remember that the data after the NULL is irrelevant – we won’t look passed that point in the array.

To set up a test for this, add this to the end of your main function:

```c
int i;
for (i = 0; i < 7; i++)
{
    add_to_list(x, 'm' + i);
}
printf("Expect to see camnopqrs: ");
print_list(x);

delete_from_list(&(x[4]));
```
printf("Expect to see camnpqrs: ");
print_list(x);

This adds some more characters to our list and then deletes one from the middle of the list. In order to get this to pass, you are going to have to write a loop in your delete function. That loop should copy a character one position to the left until it reaches the NULL. (hint: you might want to use two pointers to do this). Don’t go on until all of your tests pass.

**The Part You Build (that works magically)**

We have tests for deleting the last thing in the list and something in the middle of the list. Because we know to worry about border cases (remember that?), we should also have a test for deleting the first thing in the list. Add this to your main function:

```c
   delete_from_list(&(x[0]));
   printf("Expect to see amnpqrs: ");
   print_list(x);
```

You should be able to explain why that test should pass the first time you try it. Make sure that it really does pass!